Data Science Capstone Project- HealthCare

Problem Statement

NIDDK (National Institute of Diabetes and Digestive and Kidney Diseases) research creates knowledge about and treatments for the most chronic, costly, and consequential diseases. The dataset used in this project is originally from NIDDK. The objective is to predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Build a model to accurately predict whether the patients in the dataset have diabetes or not.

Dataset Description

The datasets consists of several medical predictor variables and one target variable (Outcome). Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and more.

Variables Description

Pregnancies -----> Number of times pregnant

Glucose -----> Plasma glucose concentration in an oral glucose tolerance test

BloodPressure -----> Diastolic blood pressure (mm Hg)

SkinThickness -----> Triceps skinfold thickness (mm)

Insulin -----> Two hour serum insulin

BMI -----> Body Mass Index

DiabetesPedigreeFunction -----> Diabetes pedigree function

Age -----> Age in years

Outcome -----> Class variable (either 0 or 1). 268 of 768 values are 1, and the others are 0

Project Task: Week 1

Data Exploration:

1. 1. Perform descriptive analysis. Understand the variables and their corresponding values. On the columns below, a value of zero does not make sense and thus indicates missing value:

• Glucose

• BloodPressure

• SkinThickness

• Insulin

• BMI

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
df= pd.read_csv('health care diabetes.csv')
```

In [3]:

```
df.head()
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [4]:

```
df.tail()
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

In [5]:

```
df.describe()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
```

memory usage: 54.1 KB

In [7]:

```
df.columns
```

Out[7]:

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [8]:

```
df.isnull().sum()
```

Out[8]:

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [9]:

```
df.nunique()
```

Out[9]:

```
Pregnancies                  17
Glucose                     136
BloodPressure                47
SkinThickness                51
Insulin                     186
BMI                         248
DiabetesPedigreeFunction    517
Age                          52
Outcome                       2
dtype: int64
```

In [10]:

```
df.shape
```

Out[10]:

```
(768, 9)
```

In [11]:

```
#Checking if there is any 0 value
df[df[['Glucose', 'BloodPressure', 'SkinThickness',
'Insulin','BMI']]==0].count()
```

Out[11]:

```
Pregnancies                   0
```

```
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

In [12]:

```
(df[df[['Glucose', 'BloodPressure', 'SkinThickness',
'Insulin','BMI']]==0].count()/len(df))*100
```

Out[12]:

```
Pregnancies                  0.000000
Glucose                      0.651042
BloodPressure                4.557292
SkinThickness               29.557292
Insulin                     48.697917
BMI                          1.432292
DiabetesPedigreeFunction     0.000000
Age                          0.000000
Outcome                      0.000000
dtype: float64
```

In [13]:

```
#Replacing 0 with the median
for i in ['Glucose', 'BloodPressure', 'SkinThickness',
'Insulin','BMI']:
    print (i,'Old Median:', df[i].median())
    Median_Value=df[df[i]!=0][i].median()
    print ('New Median',Median_Value, '\n')
    df[i].replace(0,Median_Value,inplace=True)
```

```
Glucose Old Median: 117.0
New Median 117.0

BloodPressure Old Median: 72.0
New Median 72.0

SkinThickness Old Median: 23.0
New Median 29.0

Insulin Old Median: 30.5
New Median 125.0

BMI Old Median: 32.0
New Median 32.3
```

In [14]:

```
df[df[['Glucose', 'BloodPressure', 'SkinThickness',
'Insulin','BMI']]==0].count()
```

Out[14]:

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

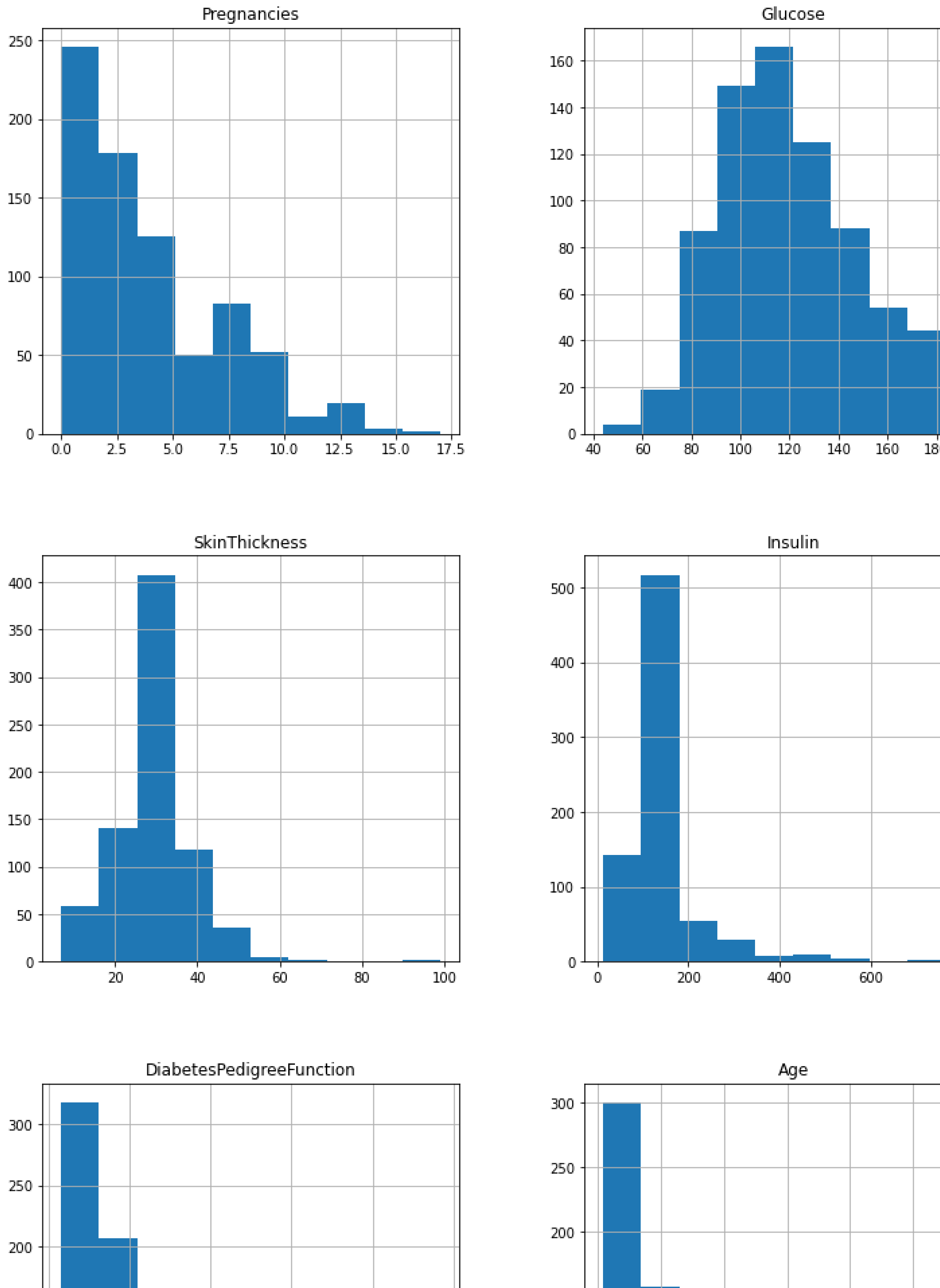
1. 1. Visually explore these variables using histograms. Treat the missing values accordingly.

In [15]:

```
df.hist(figsize=(20,20))
```

Out[15]:

```
array([[<AxesSubplot:title={'center':'Pregnancies'}>,
        <AxesSubplot:title={'center':'Glucose'}>,
        <AxesSubplot:title={'center':'BloodPressure'}>],
       [<AxesSubplot:title={'center':'SkinThickness'}>,
        <AxesSubplot:title={'center':'Insulin'}>,
        <AxesSubplot:title={'center':'BMI'}>],
       [<AxesSubplot:title={'center':'DiabetesPedigreeFunction'}>,
        <AxesSubplot:title={'center':'Age'}>,
        <AxesSubplot:title={'center':'Outcome'}>]], dtype=object)
```


Pregnancies
0
50
100
150
200
250
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Glucose
0
20
40
60
80
100
120
140
160
40
60
80
100
120
140
160
SkinThickness
0
50
100
150
200
250
300
350
400
20
40
60
80
100
Insulin
0
100
200
300
400
500
0
200
400
600
DiabetesPedigreeFunction
200
250
300
Age
200
250
300

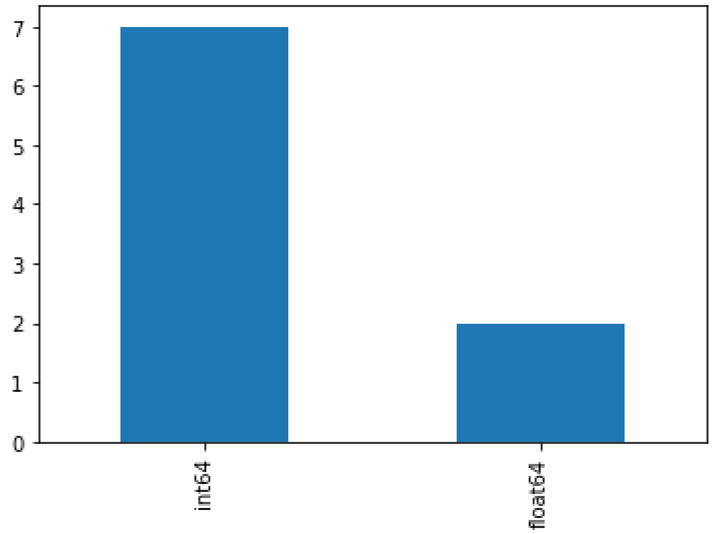
1. 1. There are integer and float data type variables in this dataset. Create a count (frequency) plot describing the data types and the count of variables.

In [16]:

```
df.dtypes.value_counts().plot(kind='bar')
```

Out[16]:

```
<AxesSubplot:>
```



Project Task: Week 2

Data Exploration:

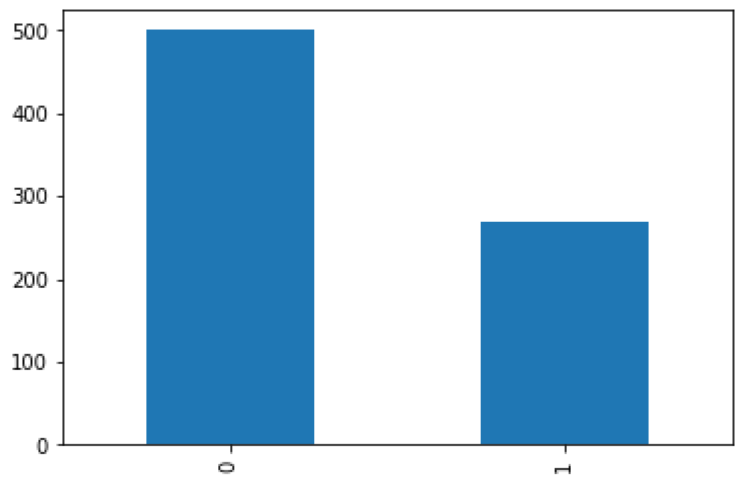
1. 1. Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action.

In [17]:

```
df.Outcome.value_counts().plot(kind='bar')
```

Out[17]:

```
<AxesSubplot:>
```



The graphs shows that the number of patients who are diabetic is half of the patients who are non-diabetic.

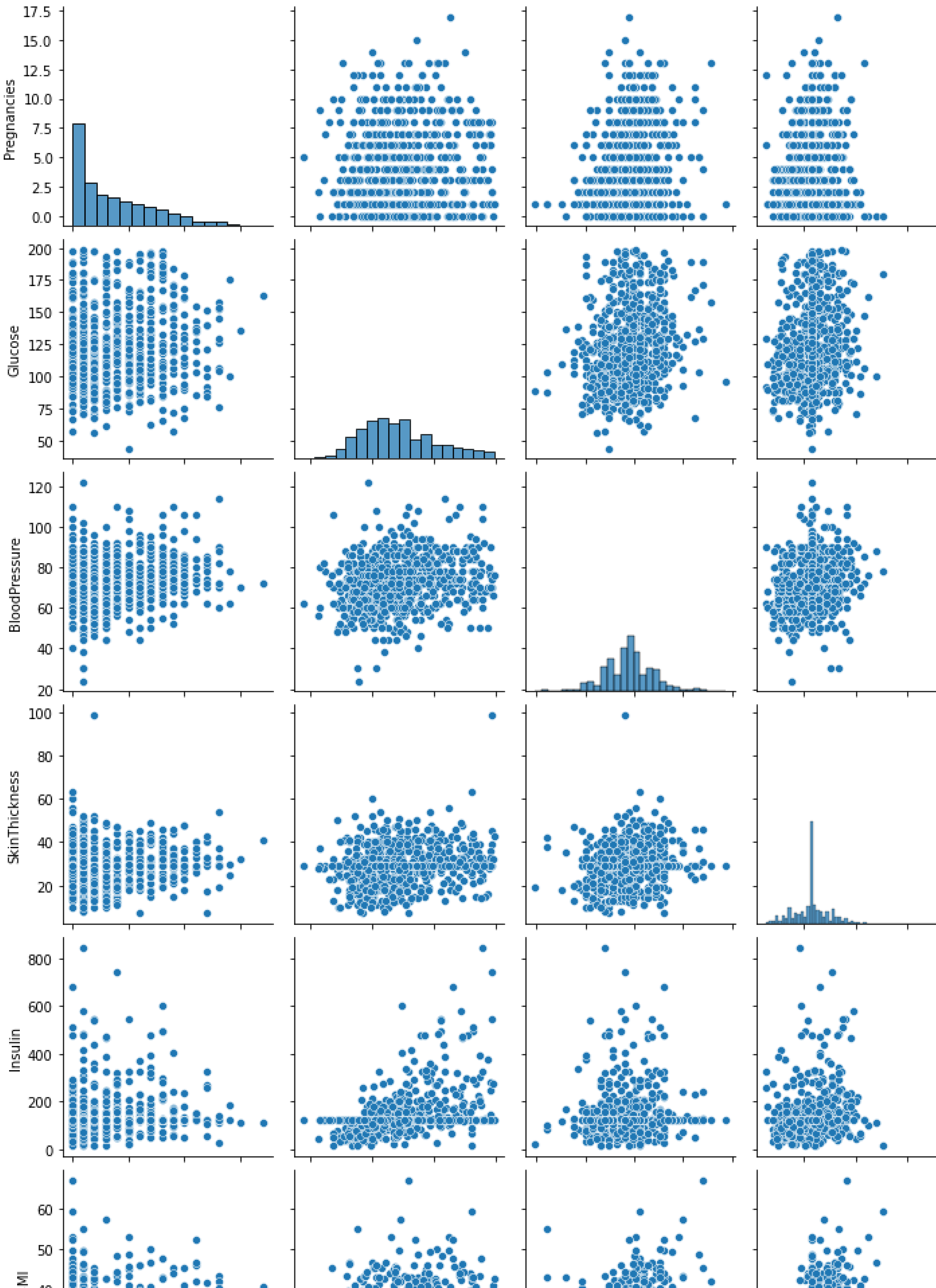
1. 1. Create scatter charts between the pair of variables to understand the relationships. Describe your findings.

In [18]:

```
sns.pairplot(df)
```

Out[18]:

```
<seaborn.axisgrid.PairGrid at 0x7f50206e8210>
```



Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI

1. 1. Perform correlation analysis. Visually explore it using a heat map.

In [19]:

```
df.corr()
```

Out[19]:

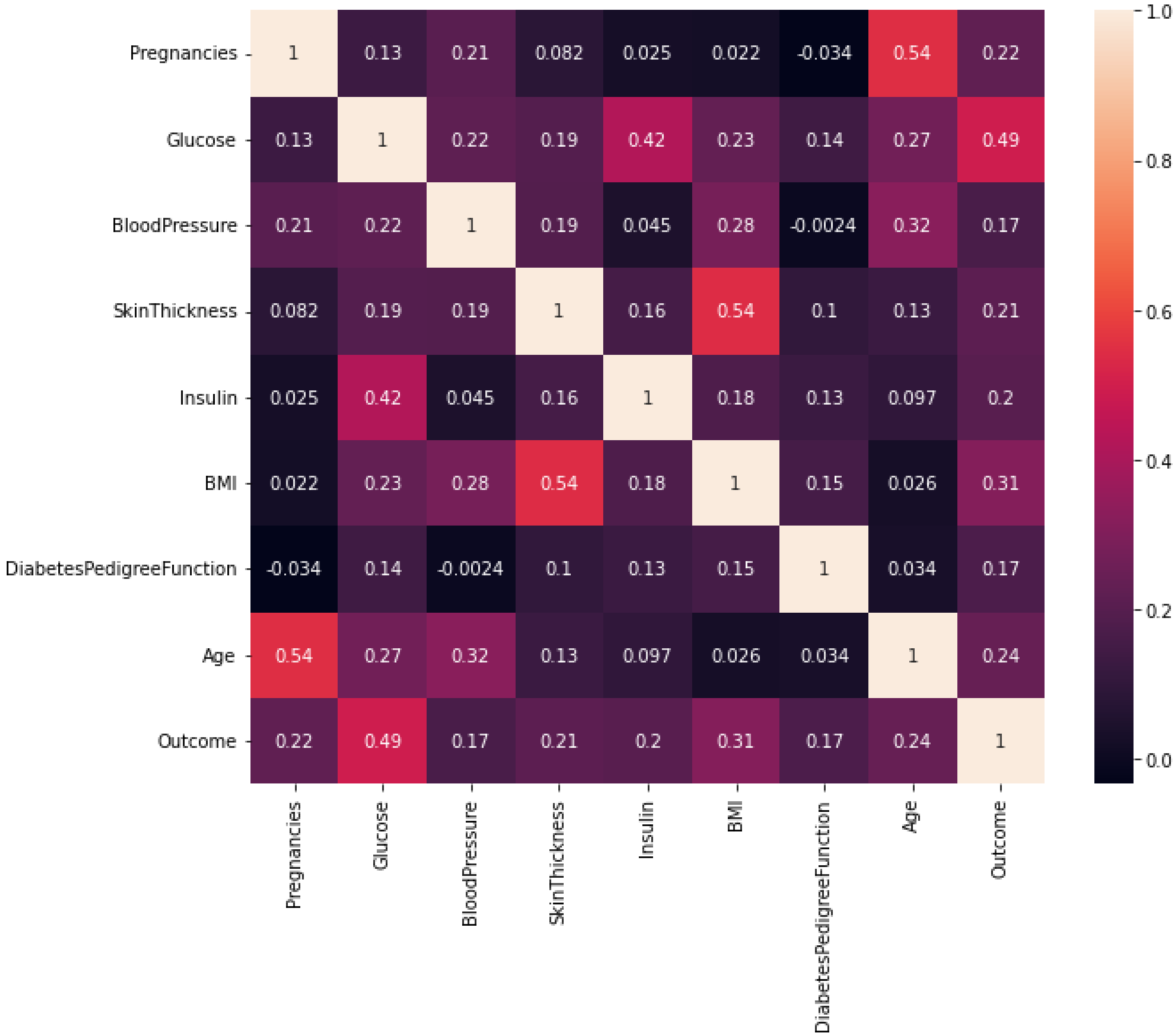
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.128213	0.208615	0.081770	0.025047	0.021559	-0.033523	0.544341	0.221898
Glucose	0.128213	1.000000	0.218937	0.192615	0.419451	0.231049	0.137327	0.266909	0.492782
BloodPressure	0.208615	0.218937	1.000000	0.191892	0.045363	0.281257	-0.002378	0.324915	0.165723
SkinThickness	0.081770	0.192615	0.191892	1.000000	0.155610	0.543205	0.102188	0.126107	0.214873
Insulin	0.025047	0.419451	0.045363	0.155610	1.000000	0.180241	0.126503	0.097101	0.203790
BMI	0.021559	0.231049	0.281257	0.543205	0.180241	1.000000	0.153438	0.025597	0.312038
DiabetesPedigreeFunction	-0.033523	0.137327	-0.002378	0.102188	0.126503	0.153438	1.000000	0.033561	0.173844
Age	0.544341	0.266909	0.324915	0.126107	0.097101	0.025597	0.033561	1.000000	0.238356
Outcome	0.221898	0.492782	0.165723	0.214873	0.203790	0.312038	0.173844	0.238356	1.000000

In [20]:

```
plt.figure(figsize = (10, 8))
sns.heatmap(df.corr(),annot = True)
```

Out[20]:

```
<AxesSubplot:>
```



Observations show that characteristics like pregnancy, glucose, skinthickness, BMI, and age are more closely associated with outcomes(Glucose as a feature is the most important in this dataset).
Project Task: Week 3

Data Modeling:

1. 1. Devise strategies for model building. It is important to decide the right validation framework. Express your thought process.

In [21]:

```
df.columns
```

Out[21]:

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin',
```

```
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [22]:

```
features = ['Pregnancies', 'Glucose', 'BloodPressure',
'SkinThickness', 'Insulin','BMI', 'DiabetesPedigreeFunction', 'Age']
x= df[features]
x
```

Out[22]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	125	33.6	0.627	50
1	1	85	66	29	125	26.6	0.351	31
2	8	183	64	29	125	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33
...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63
764	2	122	70	27	125	36.8	0.340	27
765	5	121	72	23	112	26.2	0.245	30
766	1	126	60	29	125	30.1	0.349	47
767	1	93	70	31	125	30.4	0.315	23

768 rows × 8 columns

In [23]:

```
y = df['Outcome']    #target
y
```

Out[23]:

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

In [24]:

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=
0.30,random_state=0)
```

In [25]:

```
x_train.shape
```

Out[25]:

```
(537, 8)
```

In [26]:

```
x_test.shape
```

Out[26]:

```
(231, 8)
```

In [27]:

```
y_test
```

Out[27]:

```
661    1
122    0
113    0
14     1
529    0
      ..
165    1
188    1
334    0
758    0
34     0
Name: Outcome, Length: 231, dtype: int64
```

1. 1. Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN algorithm.

Logistic Regression

In [28]:

```
from sklearn.linear_model import LogisticRegression
```

In [29]:

```
logreg = LogisticRegression()
logreg.fit(x_train,y_train)
```

Out[29]:

```
LogisticRegression()
```

In [30]:

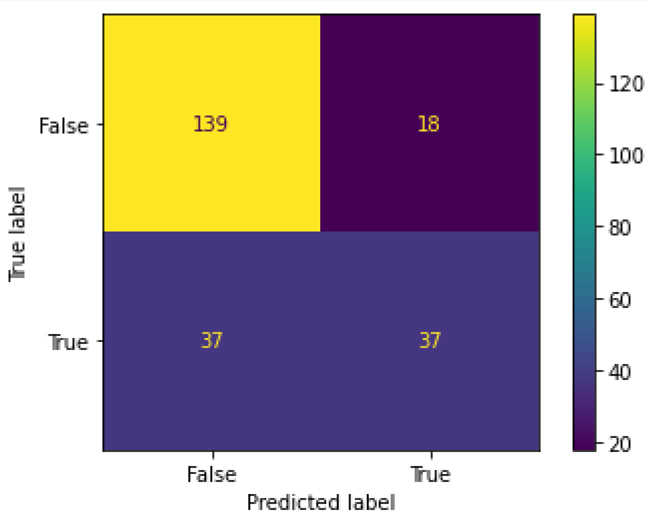
```
y_pred= logreg.predict(x_test)
y_pred
```

Out[30]:

```
array([1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1,
0,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0,
1,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
1,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
1,
       0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0,
       1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
0,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0])
```

In [31]:

```
#Evaluating the performance of the model
from sklearn import metrics
confusion_matrix= metrics.confusion_matrix(y_test,y_pred)
confusion_matrix=
metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix,
display_labels =[False,True])
confusion_matrix.plot()
plt.show()
```



In [32]:

```
#Accuracy
from sklearn.metrics import accuracy_score
```

```
accuracy_score(y_test,y_pred)
```

Out[32]:

```
0.7619047619047619
```

In [33]:

```
#Precision
from sklearn.metrics import precision_score
precision_score(y_test,y_pred)
```

Out[33]:

```
0.6727272727272727
```

In [34]:

```
#Recall
from sklearn.metrics import recall_score
recall_score(y_test,y_pred)
```

Out[34]:

```
0.5
```

In [35]:

```
#f1_score
f1_score=metrics.f1_score(y_test,y_pred)
print(f1_score)
```

```
0.5736434108527131
```

Decision Tree

In [36]:

```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
dt.fit(x_train,y_train)
```

Out[36]:

```
DecisionTreeClassifier()
```

In [37]:

```
y_pred1=dt.predict(x_test)
y_pred1
```

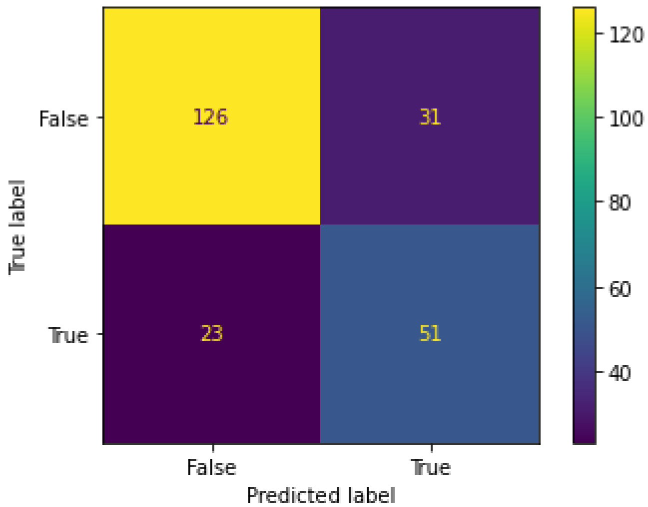
Out[37]:

```
array([1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1,
0,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0,
1,
       0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1,
1,
       0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0,
1,
```

```
       1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0,
       0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0,
       0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1,
0,
       0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
0,
       0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1])
```

In [38]:

```
from sklearn import metrics
confusion_matrix=metrics.confusion_matrix(y_test,y_pred1)
confusion_matrix=metrics.ConfusionMatrixDisplay(confusion_matrix=conf
usion_matrix, display_labels=[False,True])
confusion_matrix.plot()
plt.show()
```



In [39]:

```
#Accuracy
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred1)
```

Out[39]:

```
0.7662337662337663
```

In [40]:

```
#Precision
from sklearn.metrics import precision_score
precision_score(y_test,y_pred1)
```

Out[40]:

```
0.6219512195121951
```

In [41]:

```
#Recall
from sklearn.metrics import recall_score
recall_score(y_test,y_pred1)
```

Out[41]:

```
0.6891891891891891
```

In [42]:

```
#f1_score
f1_score=metrics.f1_score(y_test,y_pred1)
print(f1_score)
```

```
0.6538461538461539
```

Random Forest

In [43]:

```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

Out[43]:

```
RandomForestClassifier()
```

In [44]:

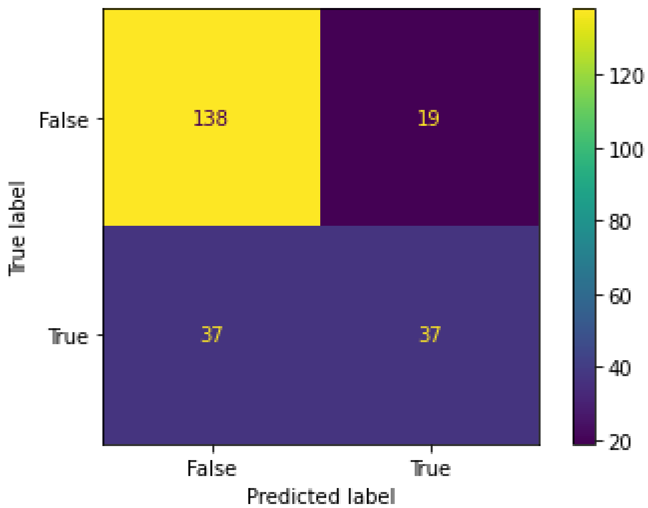
```
y_pred2=rfc.predict(x_test)
y_pred2
```

Out[44]:

```
array([1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1,
0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0,
1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1,
0,
       1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0,
       0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0,
       0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1])
```

In [45]:

```
from sklearn import metrics
confusion_matrix=metrics.confusion_matrix(y_test,y_pred2)
confusion_matrix=metrics.ConfusionMatrixDisplay(confusion_matrix=conf
usion_matrix, display_labels=[False,True])
confusion_matrix.plot()
plt.show()
```



In [46]:

```
#Accuracy
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred2)
```

Out[46]:

```
0.7575757575757576
```

In [47]:

```
#Precision
from sklearn.metrics import precision_score
precision_score(y_test,y_pred2)
```

Out[47]:

```
0.6607142857142857
```

In [48]:

```
#Recall
from sklearn.metrics import recall_score
recall_score(y_test,y_pred2)
```

Out[48]:

```
0.5
```

In [49]:

```
#f1_score
```

```
f1_score=metrics.f1_score(y_test,y_pred2)
print(f1_score)
```

0.5692307692307693

KNN

In [50]:

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=4)
knn.fit(x_train,y_train)
```

Out[50]:

KNeighborsClassifier(n_neighbors=4)

In [51]:

```
pred=knn.predict(x_test)
pred
```

Out[51]:

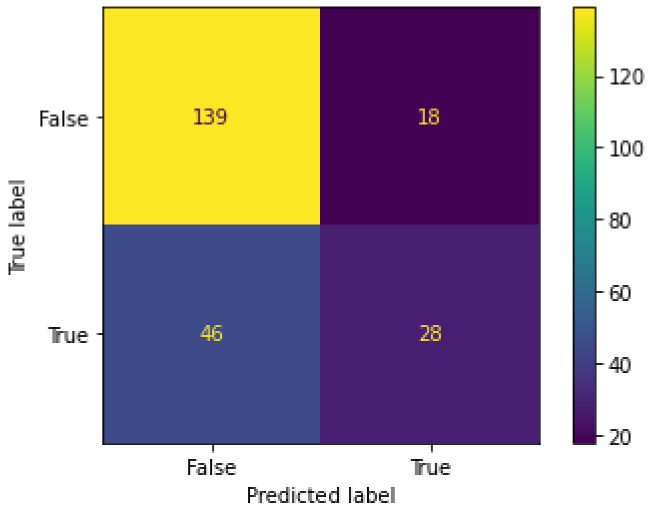
```
array([1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1,
1,
       1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
1,
       0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
0,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0])
```

In [52]:

```
from sklearn import metrics

confusion_matrix= metrics.confusion_matrix(y_test,pred)
confusion_matrix=metrics.ConfusionMatrixDisplay(confusion_matrix=conf
usion_matrix,display_labels=[False,True])
confusion_matrix.plot()
```

Out[52]:

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f50197877d0>



In [53]:

```
#Accuracy
from sklearn.metrics import accuracy_score
accuracy_score(y_test,pred)
```

Out[53]:

```
0.7229437229437229
```

In [54]:

```
#Precision
from sklearn.metrics import precision_score
precision_score(y_test,pred)
```

Out[54]:

```
0.6086956521739131
```

In [55]:

```
#Recall
from sklearn.metrics import recall_score
recall_score(y_test,pred)
```

Out[55]:

```
0.3783783783783784
```

In [56]:

```
#f1_score
f1_score=metrics.f1_score(y_test,pred)
print(f1_score)
```

```
0.4666666666666667
```

Project Task: Week 4

Data Modeling:

1. 1. Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc. Please be descriptive to explain what values of these parameter you have used.

In [57]:

```
from sklearn.metrics import classification_report, roc_curve,
roc_auc_score
```

In [58]:

```
print(classification_report(y_test,y_pred)) #Logistic Regression
```

```
    precision    recall  f1-score   support

           0       0.79      0.89      0.83       157
           1       0.67      0.50      0.57        74

    accuracy                           0.76       231
   macro avg       0.73      0.69      0.70       231
weighted avg       0.75      0.76      0.75       231
```

In [59]:

```
print(classification_report(y_test,y_pred1)) #Decision Tree
```

```
    precision    recall  f1-score   support

           0       0.85      0.80      0.82       157
           1       0.62      0.69      0.65        74

    accuracy                           0.77       231
   macro avg       0.73      0.75      0.74       231
weighted avg       0.77      0.77      0.77       231
```

In [60]:

```
print(classification_report(y_test,y_pred2)) #Random Forest
```

```
    precision    recall  f1-score   support

           0       0.79      0.88      0.83       157
           1       0.66      0.50      0.57        74

    accuracy                           0.76       231
   macro avg       0.72      0.69      0.70       231
weighted avg       0.75      0.76      0.75       231
```

In [61]:

```
print(classification_report(y_test,pred)) #KNN
```

```
    precision    recall  f1-score   support
```

```
              0       0.75      0.89      0.81       157
              1       0.61      0.38      0.47        74

       accuracy                           0.72       231
      macro avg       0.68      0.63      0.64       231
   weighted avg       0.71      0.72      0.70       231
```

Therefore Decision Tree is the best model for this prediction since it has an accuracy_score of 0.77.
For Logistic Regression

In [62]:

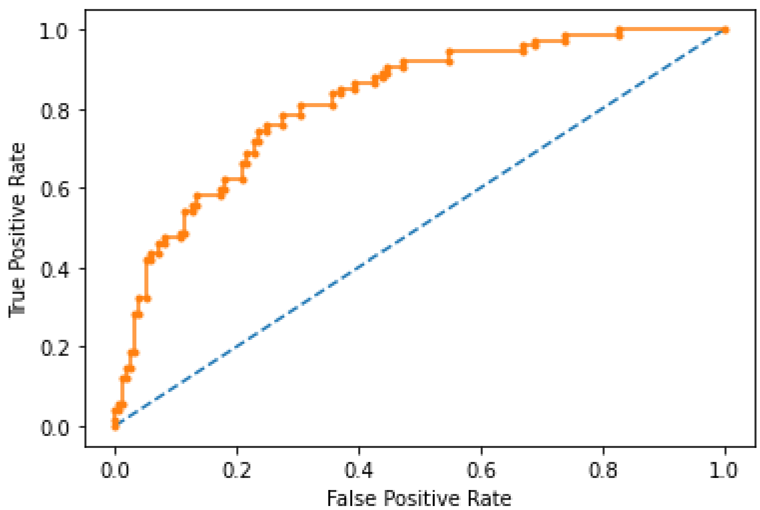
```
#Using roc_curve() to get the threshold, TPR, and FPR.
fpr,tpr,thresholds =
roc_curve(y_test,logreg.predict_proba(x_test)[:,1])
print("True Positive Rate - {}, False Positive Rate - {}, Thresholds
- {}".format(tpr,fpr,thresholds))

#For AUC using roc_auc_score()
auc1= roc_auc_score(y_test, logreg.predict(x_test))
print('AUC: %.3f' % auc1)

#Ploting the ROC curve
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr, tpr, marker='.')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

```
True Positive Rate - [0.         0.01351351 0.04054054 0.04054054
0.05405405 0.05405405
 0.12162162 0.12162162 0.14864865 0.14864865 0.18918919 0.18918919
 0.28378378 0.28378378 0.32432432 0.32432432 0.41891892 0.41891892
 0.43243243 0.43243243 0.45945946 0.45945946 0.47297297 0.47297297
 0.48648649 0.48648649 0.54054054 0.54054054 0.55405405 0.55405405
 0.58108108 0.58108108 0.59459459 0.59459459 0.62162162 0.62162162
 0.66216216 0.66216216 0.68918919 0.68918919 0.71621622 0.71621622
 0.74324324 0.74324324 0.75675676 0.75675676 0.78378378 0.78378378
 0.81081081 0.81081081 0.83783784 0.83783784 0.85135135 0.85135135
 0.86486486 0.86486486 0.87837838 0.87837838 0.89189189 0.89189189
 0.90540541 0.90540541 0.91891892 0.91891892 0.94594595 0.94594595
 0.95945946 0.95945946 0.97297297 0.97297297 0.98648649 0.98648649
 1.         1.        ], False Positive Rate - [0.         0.
0.         0.00636943 0.00636943 0.01273885
 0.01273885 0.01910828 0.01910828 0.02547771 0.02547771 0.03184713
 0.03184713 0.03821656 0.03821656 0.05095541 0.05095541 0.05732484
 0.05732484 0.07006369 0.07006369 0.08280255 0.08280255 0.10828025
 0.10828025 0.11464968 0.11464968 0.12738854 0.12738854 0.13375796
 0.13375796 0.17197452 0.17197452 0.17834395 0.17834395 0.21019108
 0.21019108 0.21656051 0.21656051 0.22929936 0.22929936 0.23566879
 0.23566879 0.24840764 0.24840764 0.27388535 0.27388535 0.30573248
 0.30573248 0.3566879  0.3566879  0.36942675 0.36942675 0.39490446
```

```
 0.39490446 0.42675159 0.42675159 0.43949045 0.43949045 0.44585987
 0.44585987 0.47133758 0.47133758 0.5477707  0.5477707  0.66878981
 0.66878981 0.68789809 0.68789809 0.7388535  0.7388535  0.82802548
 0.82802548 1.        ], Thresholds - [1.97976979 0.97976979
0.96952636 0.96465467 0.95130855 0.93167602
 0.87768516 0.86187685 0.84289126 0.8199858  0.79021783 0.77540975
 0.71885523 0.71625433 0.70931434 0.70415687 0.62141043 0.62104285
 0.61431963 0.59687457 0.57879801 0.55061878 0.54587181 0.52884655
 0.5137863  0.51014758 0.49510963 0.4862986  0.47132712 0.46968411
 0.45664249 0.43096162 0.41991735 0.41954489 0.41102134 0.39868693
 0.39323817 0.39317068 0.39003323 0.38116884 0.37154483 0.36954273
 0.35394002 0.34148568 0.34102858 0.3267451  0.31304148 0.29234675
 0.29182313 0.26468055 0.26365187 0.25865693 0.25526253 0.25129361
 0.24875032 0.24288912 0.2417289  0.23000866 0.22963736 0.22266081
 0.2222047  0.21113181 0.20706676 0.17565968 0.16860768 0.13140044
 0.13034169 0.12643356 0.12560678 0.11482271 0.11389767 0.0803826
 0.07768083 0.03802583]
AUC: 0.693
```

Out[62]:

```
Text(0, 0.5, 'True Positive Rate')
```



For Decision Tree

In [63]:

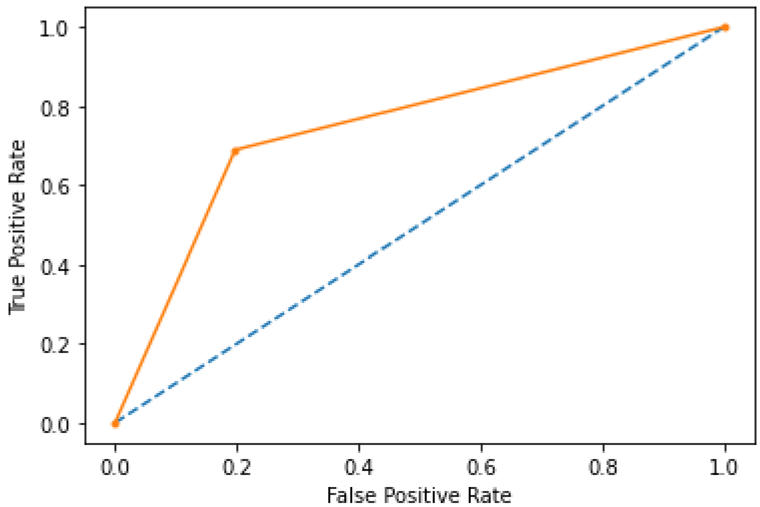
```
#Using roc_curve() to get the threshold, TPR, and FPR.
fpr,tpr,thresholds = roc_curve(y_test,dt.predict_proba(x_test)[:,1])
print("True Positive Rate - {}, False Positive Rate - {}, Thresholds
- {}".format(tpr,fpr,thresholds))

#For AUC using roc_auc_score()
auc2= roc_auc_score(y_test, dt.predict(x_test))
print('AUC: %.3f' % auc2)

#Ploting the ROC curve
```

```
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr, tpr, marker='.')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

```
True Positive Rate - [0.         0.68918919 1.        ], False
Positive Rate - [0.         0.19745223 1.        ], Thresholds - [2.
1. 0.]
AUC: 0.746
```

Out[63]:

```
Text(0, 0.5, 'True Positive Rate')
```



For Random Forest

In [64]:

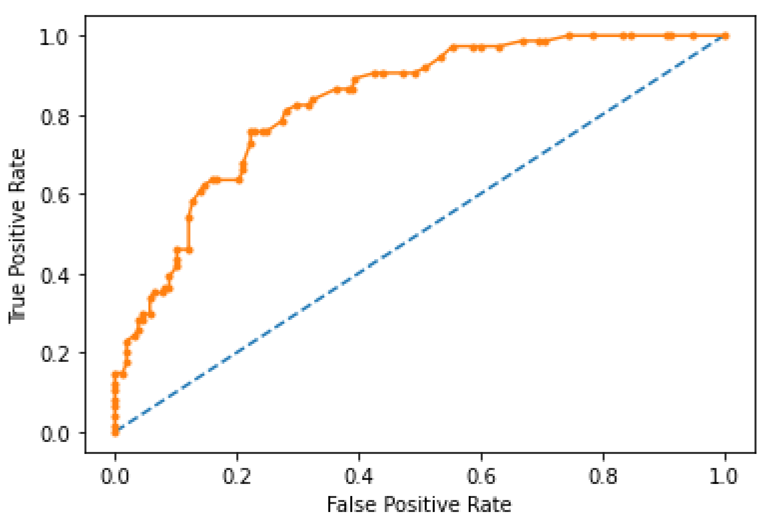
```
#Using roc_curve() to get the threshold, TPR, and FPR.
fpr,tpr,thresholds = roc_curve(y_test,rfc.predict_proba(x_test)[:,1])
print("True Positive Rate - {}, False Positive Rate - {}, Thresholds
- {}".format(tpr,fpr,thresholds))

#For AUC using roc_auc_score()
auc3= roc_auc_score(y_test, rfc.predict(x_test))
print('AUC: %.3f' % auc3)

#Ploting the ROC curve
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr, tpr, marker='.')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

```
True Positive Rate - [0.         0.01351351 0.04054054 0.06756757
0.08108108 0.10810811
 0.12162162 0.14864865 0.14864865 0.17567568 0.2027027  0.22972973
 0.24324324 0.25675676 0.28378378 0.28378378 0.2972973  0.2972973
 0.33783784 0.35135135 0.35135135 0.36486486 0.36486486 0.39189189
 0.41891892 0.43243243 0.45945946 0.45945946 0.54054054 0.58108108
```

```
 0.60810811 0.62162162 0.63513514 0.63513514 0.63513514 0.66216216
 0.67567568 0.72972973 0.75675676 0.75675676 0.75675676 0.75675676
 0.78378378 0.81081081 0.82432432 0.82432432 0.83783784 0.86486486
 0.86486486 0.86486486 0.89189189 0.90540541 0.90540541 0.90540541
 0.90540541 0.91891892 0.94594595 0.97297297 0.97297297 0.97297297
 0.97297297 0.98648649 0.98648649 0.98648649 1.         1.
 1.         1.         1.         1.         1.         1.        ],
False Positive Rate - [0.         0.         0.         0.         0.
0.
 0.         0.         0.01273885 0.01910828 0.01910828 0.01910828
 0.03184713 0.03821656 0.03821656 0.04458599 0.04458599 0.05732484
 0.05732484 0.06369427 0.07643312 0.08280255 0.08917197 0.08917197
 0.10191083 0.10191083 0.10191083 0.12101911 0.12101911 0.12738854
 0.14012739 0.14649682 0.15923567 0.1656051  0.20382166 0.21019108
 0.21019108 0.22292994 0.22292994 0.22929936 0.24203822 0.24840764
 0.27388535 0.28025478 0.29936306 0.31847134 0.32484076 0.36305732
 0.38216561 0.38853503 0.39490446 0.42675159 0.43949045 0.47133758
 0.49044586 0.50955414 0.53503185 0.55414013 0.58598726 0.59872611
 0.63057325 0.66878981 0.69426752 0.70700637 0.74522293 0.78343949
 0.8343949  0.84713376 0.9044586  0.91082803 0.94904459 1.        ],
Thresholds - [1.96 0.96 0.89 0.88 0.86 0.83 0.82 0.81 0.8  0.79 0.78
0.76 0.75 0.74
 0.73 0.72 0.69 0.68 0.67 0.66 0.65 0.64 0.63 0.62 0.57 0.55 0.54
0.53
 0.5  0.49 0.48 0.47 0.46 0.45 0.43 0.42 0.41 0.39 0.38 0.36 0.35
0.34
 0.32 0.31 0.29 0.28 0.27 0.25 0.24 0.23 0.22 0.21 0.19 0.18 0.17
0.16
 0.15 0.14 0.13 0.12 0.11 0.1  0.09 0.08 0.07 0.06 0.05 0.04 0.03
0.02
 0.01 0.   ]
AUC: 0.689
```

Out[64]:

```
Text(0, 0.5, 'True Positive Rate')
```



For KNN

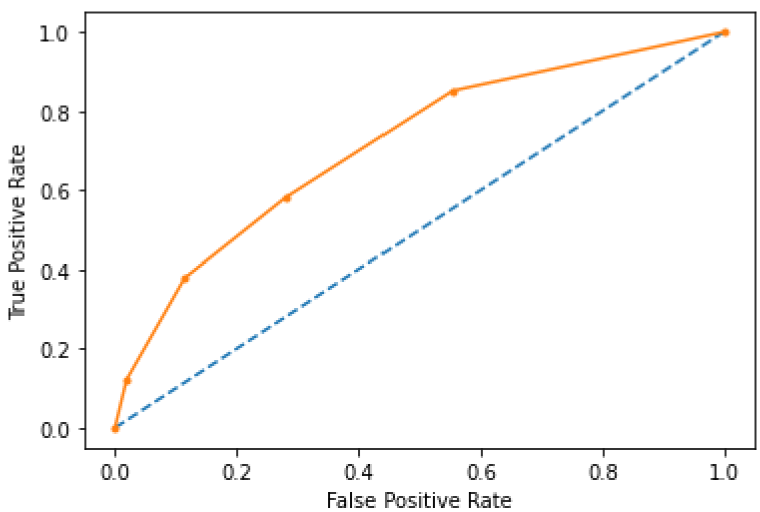
In [65]:

```
#Using roc_curve() to get the threshold, TPR, and FPR.
fpr,tpr,thresholds = roc_curve(y_test,knn.predict_proba(x_test)[:,1])
print("True Positive Rate - {}, False Positive Rate - {}, Thresholds
- {}".format(tpr,fpr,thresholds))

#For AUC using roc_auc_score()
auc= roc_auc_score(y_test, knn.predict(x_test))
print('AUC: %.3f' % auc)

#Ploting the ROC curve
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr, tpr, marker='.')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

```
True Positive Rate - [0.         0.12162162 0.37837838 0.58108108
0.85135135 1.        ], False Positive Rate - [0.         0.01910828
0.11464968 0.28025478 0.55414013 1.        ], Thresholds - [2.   1.
0.75 0.5  0.25 0.  ]
AUC: 0.632
```

Out[65]:

```
Text(0, 0.5, 'True Positive Rate')
```



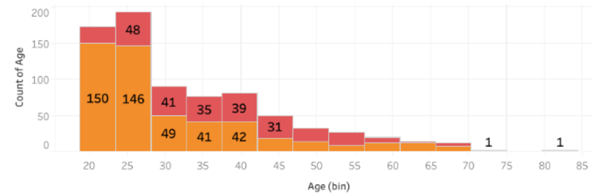
Create a dashboard

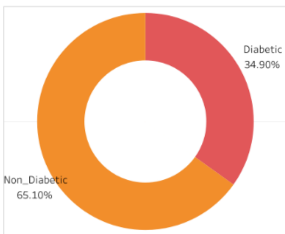
HealthCare

Correlation Analysis

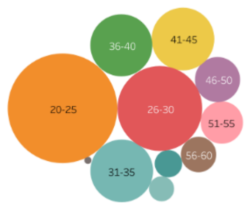
	Age Rank											
	20-25	26-30	31-35	36-40	41-45	46-50	51-55	56-60	61-65	66-70	71-75	81-85
Avg. BMI	30.4	33.0	32.8	33.0	35.3	32.9	31.8	30.2	29.9	27.5	19.6	25.9
Avg. Blood Pressure	63.8	68.0	68.8	71.8	73.3	77.9	81.9	77.5	76.0	80.7	0.0	74.0
Avg. Glucose	110.7	120.3	124.2	128.3	125.1	124.5	143.2	138.3	136.4	139.0	119.0	134.0
Avg. Insulin	84.3	84.3	92.8	59.5	56.7	67.6	109.9	149.5	26.4	0.0	0.0	60.0
Avg. Skin Thickness	22.0	21.3	20.1	21.0	18.9	20.4	16.3	18.7	20.0	1.6	0.0	33.0

Age wise Diabetic/Non-Diabetic





Bubble Chart for BP



Histogram for BP